UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Doble grado en Ingeniería Informática y Matemáticas.

TRABAJO FIN DE GRADO

SVM Ensembles for large volumes of data.

Author: David Nevado Catalán
Advisors: Gonzalo Martínez Muñoz, Alberto Suárez González

June 2019

David Nevado Catalán
SVM Ensembles for large volumes of data.

David Nevado Catalán

PRINTED IN SPAIN

Agradecimientos

En primer lugar me gustaría dar las gracias a mis tutores Gonzalo y Alberto por sus valiosos consejos y su gran ayuda en este proyecto. Este trabajo no sería lo que es de no ser por su dedicación e implicación permanentes. También me gustaría dar las gracias Centro de Computación Científica (CCC) de la UAM por permitirnos utilizar sus instalaciones.

A mis compañeros de trabajo Iñigo y Jorge: por compartir su experiencia y conocimientos conmigo y enseñarme tanto acerca de Machine learning; y por supuesto a mis compañeros de universidad, que me han acompañado durante estos últimos 5 años y han compartido conmigo los mejores y los peores momentos de la carrera.

Por último, y sobre todo, quiero dar las gracias a mi familia: mis padres y mi hermana, por su apoyo constante durante toda la carrera, y en particular durante la realización de este trabajo.

Resumen

Las Máquinas de Vectores de Soporte o SVM (*Support Vector Machines*) por sus siglas en inglés, son un algoritmo de de machine learning (ML) ampliamente utilizado por su excelente rendimiento en problemas de clasificación y regresión. No obstante, su alto coste computacional es frecuentemente un factor limitante para su aplicación, especialmente en problemas que tratan con grandes volúmenes de datos.

En este trabajo de fin de grado hemos diseñado, desarrollado y analizado un modelo de ML basado en ensembles de SVM, especialmente enfocado a problemas con grandes datasets. Para alcanzar este objetivo combinamos varias técnicas de *Ensemble methods* e introdujimos una variación de *Subbagging* especialmente diseñada para aprovechar el gran volumen de datos disponible en estos problemas. El modelo resultante muestra muy buen rendimiento en comparación con otros con modelos: Comparado con otros ensembles de SVM con los mismos requisitos computacionales el ensemble desarrollado tiene una mayor estabilidad en sus puntuaciones. Comparado con una sola SVM el modelo propuesto alcanza mayores puntuaciones para un mismo tiempo de entrenamiento. Además, alcanza puntuaciones equivalentes a una sola SVM entrenada sin limitaciones de tiempo, usando solo un 10% de su tiempo de entrenamiento, incluso menos en algunos casos.

Palabras clave

Inteligencia artificial, Aprendizaje atuomático, Máquina de vectores de soporte, SVM, Métodos de ensemble

Abstract

Support Vector Machines (SVM) are a popular machine learning (ML) algorithm that has been extensively used for its remarkable performance in tasks of classification and regression. However, its high computational complexity is often a limiting factor for its use, specially in the context of problems with large volumes of data. The extraordinary increase in the availability of data experienced in the last decades demand the development of new algorithms in the field of ML that are able to deal with this ever increasing volumes of data.

In this undergraduate thesis we have designed, developed and analyzed a ML model based on SVM ensembles specially aimed at problems with large datasets. To achieve this goal we combined several ensemble methods and introduced a modified version of subbagging that capitalized on the high availability of data. The resulting model shows a very good performance in comparison to other models: Compared to other SVM ensembles with equal computational requirements the developed ensemble achieves greater stability in its scores. Compared to a single SVM the proposed model reaches higher accuracies for the same training time budget. Furthermore, it achieves comparable accuracies to a single SVM trained without time limitations, using only a 10% of its training time, even less in the best cases.

Keywords

Artifical Intelligence, Machine Learning, Support Vector Machine, SVM, Ensemble learning

Table of Contents

Lists

List of figures

List of tables

1 Introduction

In the last decades, the advances made in the fields of telecommunications and computer science have dramatically increased the availability of data. For machine learning, these advances come with the challenge of developing new systems that are able to take advantage of these ever increasing volumes of data.

Support Vector Machines (SVM) were introduced in 1995 by C.Cortes and V.Vapnick in their article *Support Vector Networks* [1]. Almost 25 years later it is still a popular machine learning algorithm used in a wide range of applications [2] for its strong theoretical foundations and its remarkable performance. However, despite the introduction of some optimizations and improvements [3], it is still a complex algorithm and its computational cost quickly becomes prohibitive when the volume of data used increases. In this work, we leverage on ensemble methods [4] to develop a model based on the SVM algorithm that has a low computational cost so it may be suited for problems with large datasets.

1.1 Goals and scope of the project

In this work we design and develop a machine learning algorithm combining SVM and Ensemble methods. Our main goal is to achieve the high performance and stability of a regular SVM while significantly reducing its computational cost. In particular, we will focus on problems where large volumes of data are available.

This work is limited to the binary classification task. However, we hope that the results and insights obtained can be extended to problems of non-binary classification and regression.

1.2 Document structure

Besides the introduction, this document is composed of four chapters and one appendix. In Chapter 2 we review the state of the art and make an in depth exposition of the technologies most relevant to this work: Support Vector Machines and machine learning ensemble methods.

In Chapter 3 we present the model developed in this project along with its most relevant variations. We also present the key aspects of its implementation and development process.

In Chapter 4 we analyze the performance of the proposed model. We do so by presenting 3 tests designed to evaluate different aspects or our model. In each of these tests we start by defining the motivation of the test and the aspect to be evaluated. Then we describe in detail the methodology used in order to make our results reproducible. Finally we present the results obtained and discuss their implications.

To conclude, Chapter 5 summarizes the fundamental ideas used in the design of the model and presents the most relevant results and conclusions drawn from the tests. In this chapter we also suggest some of the lines along which this work could be continued.

Appendix A contains the pseudocode for some of the tests carried out in Section 4.2.

2

State of the art

Machine learning (ML) is a branch of computer science that allows computer systems to improve or learn from experience. This experience typically consists in the observation of data and the learning process is the identification of patterns and trends on this data. With these patterns ML systems acquire the ability of making predictions on new data [5]. ML algorithms can be classified in two categories depending on the type of data they use: Supervised learning and unsupervised learning. In supervised learning the training data fed to the algorithm includes the desired solutions. These solutions are commonly referred as labels and so, we say supervised learning deals with labeled data. Two typical tasks of supervised learning are classification and regression. In unsupervised learning the data for the algorithm is unlabeled. Typical unsupervised learning tasks are clustering, which consists in inducing plausible partitions of the data; dimensionality reduction, whose goal is to simplify the data minimizing the loss of information; and anomaly detection, which consists in the detection of outliers.

In this work we will focus on supervised learning, particularly, on the classification task. In this task the goal is to predict a discrete class label based on a vector of features. A classic example of classification problem is given by Fisher's Iris dataset [6]. In this problem the goal is to predict a flower category based on some of its features such as petal length and width. There have been many ML algorithms proposed for classification. Some of the most common are logistic regressors, k-nearest neighbors (k-NN) [7], support vector machines (SVM) [2, 8], neural networks (NN) [9], decision trees (DT) [10], and ensemble methods [11]. The goal of this project is to develop a classification model based on SVMs and ensemble methods that maintains their qualities while mitigating their drawbacks.

In this chapter we will make an exposition of the most relevant topics to this work. In section (2.1) we will present the ML problem of classification in greater detail. The subsequent sections (2.2, 2.3) are dedicated to the models used to develop our classifier: SVM and Ensemble methods.

2.1 Classification

Classification is a type of supervised leaning problem in which a class label is predicted from vector of features. Let us consider a training set $\mathcal{D}_{train} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{train}}$ formed by N_{train} instances. For an instance

$(\mathbf{x}_i, y_i)$ we call $\mathbf{x}_i$ the feature vector or attribute vector and y_i the class label. The feature vectors are elements of the feature space $\mathcal{X}$, and the labels are elements of the label space $\mathcal{Y}$, which is a discrete space comprised of all possible class labels. Therefore, in a binary classification problem, $\mathcal{Y}$ is a two element space, and can be encoded as $\mathcal{Y} = \{-1, 1\}$. Throughout this chapter we will assume that the instances in the datasets are independent and identically distributed (iid) random variables sampled from a fixed underlying distribution $\mathcal{S}$ defined over $\mathcal{X} \times \mathcal{Y}$.

The goal in a classification problem is to obtain a function or classifier $c : \mathcal{X} \to \mathcal{Y}$ that maps the feature vector of an instance $\mathbf{x}$ to its corresponding label y. To evaluate the performance of this function we define its generalization error

$$Error(c) := Pr_{(\mathbf{x},y)\sim\mathcal{S}}[c(\mathbf{x}) \neq y]. \tag{2.1}$$

It is defined as the probability of misclassifying an instance $(\mathbf{x}, y)$ drawn at random form $\mathcal{S}$. Computing the generalization error requires knowing the underlying distribution $\mathcal{S}$. In some special cases, such as artificial problems with synthetic data, we have access to this distribution, but generally S is unknown. For these cases we can estimate the generalization error by using a test set $\mathcal{D}_{test} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{test}}$ independent from $\mathcal{D}_{train}$ with the same underlying distribution $\mathcal{S}$. The test error is defined as

$$Error_{test}(c) := \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \mathbb{1}[c(\mathbf{x}_i) \neq y_i], \tag{2.2}$$

where $\mathbb{1}$ represents the indicator function, taking value 1 if the condition is true and 0 otherwise.

There is a lower bound for the generalization error in any given problem. This bound is known as Bayes error and it is defined as the error of the optimal classifier c^*.

$$c^* := \arg\max_{y\in\mathcal{Y}} P(y|\mathbf{x}) \tag{2.3}$$

As with the generalization error, Bayes error can be calculated only when the underlying distribution $\mathcal{S}$ is known.

The generalization error is often expressed as the sum of two terms: Bias and Variance. [12] This decomposition is useful to analyze the performance of predictors. Let $\mathcal{L}$ be the learning algorithm used to get a predictor or classifier from the data:

$$\mathcal{L}(\mathcal{D}_{train}) = c \tag{2.4}$$

In order to differentiate bias and variance it is necessary to define first the central tendency. The central tendency is the most probable label for a given instance $\mathbf{x}$ and is defined as

$$c^o_{\mathcal{D}_{train}} = \arg\max_{y\in\mathcal{Y}} P_{D_{train}}(y|\mathbf{x}) \tag{2.5}$$

The bias is the deviation of the central tendency from the actual labels of the instances.

$$bias = P_{\mathcal{S},\mathcal{D}_{train}}(\mathcal{L}(\mathcal{D}_{train})(\mathbf{x}) \neq y) \wedge \mathcal{L}(\mathcal{D}_{train})(\mathbf{x}) = c^{o}_{\mathcal{L},\mathcal{D}_{i}rain}(\mathbf{x}) \tag{2.6}$$

The variance is the deviation of the predictions from the central tendency.

$$variance = P_{\mathcal{S},\mathcal{D}_{train}}(\mathcal{L}(\mathcal{D}_{train})(\mathbf{x}) \neq y) \wedge \mathcal{L}(\mathcal{D}_{train})(\mathbf{x}) \neq c^{o}_{\mathcal{L},\mathcal{D},rain}(\mathbf{x}) \tag{2.7}$$

The SVM is a popular algorithm used for classification problems. It is a model with solid theoretical foundations and has proven to be a strong and reliable classifier in a wide range of applications [8, 13]. In the next section we present the key concepts of the SVM.

2.2 Support Vector Machines

Support Vector machines (SVM) were introduced 25 years ago [1, 14] and have become popular for their accuracy and robustness in problems of classification and regression [2]. We will limit this exposition to classification, which is the focus of this work.

Let us consider a classification problem where we are given a dataset $\mathcal{D}$ with N samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. $\mathbf{x}_i$ represent the input feature vectors and y_i represent the target values or labels. In this case we will consider a binary classification problem, so we will have two possible class labels. In a SVM the classification of a new instance is done by determining the sign of a certain function. For this reason, it is particularly convenient to encode the labels with the values $\{-1, 1\}$.

The key idea of the model is the use of a fixed feature-space transformation ϕ of the attributes. We write

$$\begin{aligned} \phi : \mathcal{X} &\longrightarrow \quad \tilde{\mathcal{X}} \\ \mathbf{x} &\longmapsto \phi(\mathbf{x}) \end{aligned}$$

where $\mathcal{X}$ is the original feature space and $\tilde{\mathcal{X}}$ is the transformed feature space.

We will first explain a simple scenario in which the data are linearly separable in the transformed feature space $\tilde{\mathcal{X}}$. Later we will consider a more realistic situation where this is not the case and we have overlapping class distributions and thus, the data are non-linearly separable.

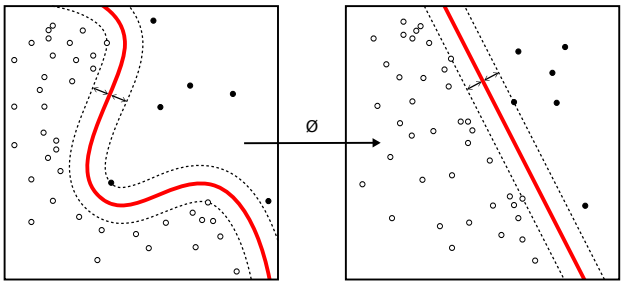

Figure 2.1: Feature space transformation of a non linear problem. The decision boundary is marked in red and the maximum margins are represented dotted lines. Original file by Alisneaky, svg version by User:Zirguezi License: CC BY-SA 4.0

2.2.1 Linearly separable case

For now, we assume the transformation ϕ converts the original problem into a linear separable one (see Figure 2.2.1) that can be solved with a model of the form

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b. \tag{2.8}$$

We can find $\mathbf{w}, b$ such that the equation (2.8) satisfies $f(\mathbf{x}_i)y_i > 0$ for all $i = 1, \cdots, N$. However, the solution is probably not unique. In this case we should aim for the solution that provides the smallest generalization error. Support Vector Machines address this challenge by introducing the concept of *margins*. The margin is defined as the minimum distance between the decision boundary and any of the training samples. SVM are maximum margin classifiers, this means that from all the valid solutions they aim for the one which provides the maximum margin. The reason for this is that wider margins provide greater generalization capacity and therefore, a better model. By using equation (2.8) we can see that the the distance between any given point $\mathbf{x}$ and the decision boundary is given by $\frac{|f(\mathbf{x})|}{||\mathbf{w}||}$. Notice that the points $\mathbf{p}$ in the decision boundary satisfy $f(\mathbf{p}) = 0$. Hence, the margin is

$$\min_{\mathbf{x} \in \mathcal{D}} \left\{ \frac{|f(\mathbf{x})|}{||\mathbf{w}||} \right\}. \tag{2.9}$$

Since we have $t_i \cdot f(\mathbf{x}_i) > 0$ for all i we can rewrite equation (2.9) as:

$$\min_{\mathbf{x} \in \mathcal{D}} \left\{ \frac{t \cdot f(\mathbf{x})}{||\mathbf{w}||} \right\}. \tag{2.10}$$

Thus the maximum margin solution is found by solving:

$$\arg\max_{\mathbf{w}, b} \left\{ \frac{1}{||\mathbf{w}||} \min_n [t_n \cdot (\mathbf{w}^T \phi(\mathbf{x}_n) + b)] \right\}. \tag{2.11}$$

In order to solve this optimization problem, we will transform it into a simpler one. First, we rescale the parameters $\mathbf{w}, b$ by a constant factor so that for the closest point to the separating hyperplane we have

$$t_n \cdot (\mathbf{w}^T \phi(\mathbf{x}_n) + b) = 1. \tag{2.12}$$

It is important to notice that this does not change distance between a point and the decision boundary defined earlier. As a consequence of (2.12) we have

$$t_n \cdot (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 \tag{2.13}$$

for all points $i = 1 \dots N$. The instances where the equality holds are known as *active constraints* whereas the others are known as *inactive constraints*. The active constraints are also called support vectors. As there is always a closest point to the decision boundary, there is always at least one active constraint. When the optimization is finished there will be at least two of them.

The optimization problem now boils down to maximizing $\|\mathbf{w}\|^{-1}$, which is equivalent to minimizing $\|\mathbf{w}\|^2$. We are left with a constrained optimization problem:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.14}$$

with the N constraints from equation (2.13). The factor $\frac{1}{2}$ has been added for later convenience. Now to solve this constrained optimization problem one simply introduces Lagrange multipliers $\mathbf{a} = (a_1, \dots, a_N)$, one per each constraint in 2.13. We obtain the Lagrangian:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n\{t_n \cdot (\mathbf{w}^T \phi(x_n) + b) - 1\}. \tag{2.15}$$

Now we minimize the Lagrangian with respect to $\mathbf{w}$ and $\{a_n\}_{n=1}^N$. By setting the derivatives of the Lagrangian with respect to $\mathbf{w}$ and b to 0 we get conditions:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \tag{2.16a}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{2.16b}$$

Now we introduce the kernel function as:

$$k(\mathbf{x}_i, \mathbf{x}_j) := \phi(\mathbf{x}_i^T)\phi(\mathbf{x}_j). \tag{2.17}$$

Using this definition and conditions (2.16a, 2.16b) we reformulate this convex constrained optimization problem to obtain its dual representation [5]. We obtain a new Lagrangian

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{2.18}$$

with constraints

$$a_n \geq 0 \quad \forall n \tag{2.19a}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{2.19b}$$

The dual representation transforms the original optimization problem (2.14), which is defined over M variables (M being the dimension of the transformed feature space $\tilde{X}$) into an optimization problem over N variables. This allows for transformations where the feature space dimensionality greatly exceeds the number of samples. It even opens up the possibility for infinite feature spaces.

In either representations, the solution is found by solving a quadratic programming problem. The complexity of these problems over N variables is $O(N^3)$. However in practice thorough computational methods and the Sequential Minimal Optimization (SMO) algorithm times of $O(N^2)$ are achieved on average [3].

In order to classify new instances **x**, we evaluate the sign of the function $f(\mathbf{x})$ defined in (2.8). We can substitute **w** in this equation using equality (2.16a), arriving to an expression in terms of the vector of Lagrange multipliers **a** and the kernel function k instead.

$$f(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{2.20}$$

To determine the sign of $f(\mathbf{x})$ we would require N evaluations of the kernel function. Luckily, we can use a property of this type of optimizations problems to drastically reduce the number of such evaluations. This optimization problem satisfies the *Karush-Kuhn-Tucker* (KKT) constraints:

$$a_n \geq 0 \tag{2.21a}$$

$$t_n f(\mathbf{x}_n) - 1 \geq 0 \tag{2.21b}$$

$$a_n(t_n f(\mathbf{x}_n) - 1) = 0. \tag{2.21c}$$

Then, for each point $\mathbf{x}_i$ we have $a_i = 0$ or $t_i f(\mathbf{x}_i) = 1$. For all points satisfying $a_i = 0$ its correspondent summand $a_i t_i k(\mathbf{x}, \mathbf{x}_i)$ in (2.20) is 0 and thus, can be removed. By removing these points we are left with a subset of the data whose points satisfy $t_i f(\mathbf{x}_i) = 1$ and therefore, lie on the maximum margins of the decision boundary. These are the *support vectors*. Once the training phase is completed these are the only points that influence the classification of a new instance and thus, all the rest can be discarded.

2.2.2 Non-linearly separable case

We will consider now a scenario where the classes in our classification problem have overlapping distributions. In this case we may find the training data points are not separable even in the transformed feature space $\tilde{X}$ or that, if they are separable, an exact separation leads to a solution with poor generalization.

To address this problem we reformulate our optimization problem. We do so by defining an error function that allows points to be classified in the wrong side of the boundary, but with a penalty that increases with their distance to this boundary.

To this end, we introduce slack variables, ζ_i $i = 1, \dots N$ for each training data point. These variables will be set $\zeta_n = 0$ if its correspondent point $\mathbf{x}_i$ is correctly classified and beyond the hyperplanes that delimit the margin, and $\zeta_i = |t_i - f(\mathbf{x}_i)|$ if they are misclassified. By introducing this variables we can transform the original constraints (2.13) into:

$$t_i f(\mathbf{x}_i) \geq 1 - \zeta_i \quad \text{with} \quad \zeta_i \geq 0 \qquad i = 1, \dots, N. \tag{2.22}$$

We now have a system with more variables but with a set of inequalities with which is is easier to work. The optimization objective function becomes:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{n=1}^{N} \zeta_n. \tag{2.23}$$

which is the original function with an added error term. With this formulation a new parameter $C > 0$ is introduced, which controls the impact of the error term in the overall function. This parameter can be seen as the inverse of a regularization coefficient. It controls the trade-off between minimizing the training error and the model complexity. When $C \to \infty$ penalization of misclassified instances increases to the point they are not allowed, effectively resulting in the model presented at the beginning for the linearly separable case. In practice, appropriate tuning of this hyperparameter is critical. If C is too high, the model will overfit the data resulting in narrow margins and poor generalization capacity. If C is too low then the model may not have enough expressive capacity to find the optimal solution, this is known as underfitting and also results in poor performance of the model.

Optimization of this problem is a bit more complex than in our previous scenario but completely analogous. Introducing Lagrange multipliers $\mathbf{a} = (a_1, \cdots, a_N)$ and $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_N)$ the Lagrangian now is:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|w\|^2 + C \sum_{n=1}^{N} \zeta_n - \sum_{n=1}^{N} \{t_n f(\mathbf{x}_n) - 1 + \zeta_n\} - \sum_{n=1}^{N} \mu_n \zeta_n \tag{2.24}$$

and the KKT constraints are:

$$a_n \geq 0 \tag{2.25a}$$

$$\mu_n \geq 0 \tag{2.25b}$$

$$\zeta_n \geq 0 \tag{2.25c}$$

$$\mu_n \zeta_n = 0 \tag{2.25d}$$

$$t_n f(\mathbf{x}_n) - 1 + \zeta_n \geq 0 \tag{2.25e}$$

$$a_n(t_n f(\mathbf{x}_n) - 1 + \zeta_n) = 0 \tag{2.25f}$$

Setting the derivatives respect **w**, b and $\{\zeta_n\}$ to 0 and replacing $f(\mathbf{x}_n)$ using (2.8) we get:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \tag{2.26a}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{2.26b}$$

$$a_n = C - \mu_n \tag{2.26c}$$

These equations can be used to eliminate **w**, b and $\{\zeta_n\}$ from the Lagrangian. This allows us to obtain the dual representation the convex constrained optimization problem with a new Lagrangian

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{2.27}$$

Notice that it is identical to the separable case, but now the constraints are different. All that is left is minimizing $\widetilde{L}$ with respect to **a** subject to the *box contraints*:

$$0 \leq a_n \leq C \tag{2.28a}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{2.28b}$$

The method for classifying a new instance is identical to the one presented (2.20). In the same way, a subset of the training data will have $a_n = 0$. These data points do not contribute the prediction and therefore are discarded, leaving only the support vectors. Support vectors satisfy $a_n > 0$ and therefore we have

$$t_n f(\mathbf{x}_n) = 1 - \zeta_n. \tag{2.29}$$

This allows for the following classification of the support vectors: If $a_n < C$, by (2.26c) we have $\mu_n > 0$ and by (2.25d) $\zeta_n = 0$. These points satisfy $t_n f(\mathbf{x}_n) = 1$ and therefore lie on the margin. If $a_n = C$ we get $t_n f(\mathbf{x}_n) \leq 1$, these points can lie also inside the margin. They are correctly classified if $\zeta_n < 1$ and misclassified if $\zeta_n > 1$.

In this work our goal is to build SVM ensembles that are at least as accurate a single SVM at a reduced computational cost. However, building SVM ensembles is a challenging task. SVM are strong and stable classifiers, and for this reason, they are difficult to diversify without reducing their accuracy. In the next section we will present the most common ensemble methods used in ML.

2.3 Ensemble methods

In machine learning, an ensemble is model formed by a collection of predictors or base learners, whose output is combined in some way to produce a joint decision. The main idea behind ensemble methods is to take advantage of the diversity of the base predictors in a way that they complement each other's decisions. It has been shown that combining several predictors in an ensemble often leads to better performance than a single ensemble predictor both in classification and regression problems. [5, 15] Nonetheless, for an ensemble to be effective its individual classifiers have to make different predictions [16, 17]. Ideally, the errors of each ensemble predictor are independent, so that they averaged in the combined decision improving the performance of the ensemble.

Consider a binary classification task and an ensemble E formed by M classifiers $\{c_1, \ldots, c_M\}$. Let the prediction of the ensemble E be the most common label predicted by the base classifiers c_m. This is just one way of combining the base learners predictions. Other are possible and some of them will be presented later in this chapter. With this strategy we have

$$E(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{m=1}^{M} \mathbb{1}[c_m(\mathbf{x}) = y], \tag{2.30}$$

where $\mathbb{1}$ represents the indicator function, taking value 1 if the condition is true and 0 otherwise.

If all classifiers make identical predictions then the ensemble E performs exactly as any of the base predictors individually, and therefore it cannot improve the accuracy over the base learners. By contrast, if the predictions $\{c_m(\mathbf{x})\}_{m=1}^{M}$ are independent then the ensemble will correctly classify $\mathbf{x}$ if at least half of the individual classifiers do so. Let p_x be the estimated error of each base classifier, the ensemble error is given by the expression

$$Error(E) = \sum_{m=\lfloor M/2 \rfloor + 1}^{M} \binom{M}{m} p_x^m (1 - p_x)^{M-m}, \tag{2.31}$$

which is simply $Pr(B > M/2)$ where B is a binomial random variable with parameters (p, M).

From this expression we observe that if the base classifiers make predictions better than random guesses ($p_x > 1/2$) and their errors are uncorrelated then $Error(E) < p_x = Error(c_m)$ with $M > 1$. The ensemble then improves the accuracy of any individual classifier. In fact, we have $Error(E) \xrightarrow{M \to \infty} 0$ as a consequence of the Condorcet's Jury Theorem [18].

Depending on whether base learners of an ensemble are the of the same type or of different types we distinguish two groups: Homogeneous ensembles and heterogeneous ensembles. In homogeneous ensembles all the base predictors are of the same kind (SVMs, Neural Networks, etc.) Random forests [11] are a common example of homogeneous ensemble that use decision trees (DTs) as base classifiers. Heterogeneous ensembles are composed of different kinds of predictors. This difference can lead to different biases in the base predictors and this in turn, to a better performance of the en-

semble.

We will present some strategies for building homogeneous ensembles. As we saw previously, diversity is key in order to improve the accuracy of the ensemble. Introducing this diversity in the base learners is generally done using two mechanisms.

- Generating altered versions of the training set for each of the base classifiers to be trained on.
- Introducing changes in the learning algorithms of the base classifiers.

One way of building different versions of the training set is generating bootstrap sets sampling from the original. The most common variants are:

1.– Resampling with replacement. This technique generates versions of the training set with repeated instances. Bagging is a popular method that uses this approach [4].

2.– Resampling without replacement. This technique does not repeat instances, therefore it must generate bootstrap sets with less instances than the original train set in order for them to be different. Subbagging is a popular method that uses this approach [19].

3.– Weighted resampling. In this technique the samples are drawn from the train set taking into account the weights of each instance. The instances with higher weights have higher probability of being selected. In this technique samples can be drawn with or without replacement. Boosting is a popular method that uses this approach [12].

Another approach used for generating train set variations is modifying the features, labels or weight of the original instances.

1.– Altering features. This technique consists of training each individual predictor using only a certain subset of features from the original feature space. When the features for the subspace are chosen at random this technique is referred as *Attribute Bagging* [20]. This approach is specially useful for data with redundant features.

2.– Altering class labels. Class labels can be switched at random or using some special criteria in order to get variations of the train set. This technique has proven to be specially useful for data subject to class label noise [21].

3.– Modifying the weights of the instances. Instances can be assigned different weights in order to modify their influence on the learning algorithm. Boosting [22] is a good example of this technique. Initially all instances are given the same weight and as the algorithm progresses these weights are modified giving more relevance to previously misclassified instances.

The second of the previously mentioned strategies for introducing diversity in an ensemble: Modifying the learning algorithm; can be applied in several ways.

1.– Different hyperparameters. A variation in the model hyperparameters will lead to different predictors. For example, for a SVM, ensemble kernels and regularization parameters for the individual learners can be altered to achieve a diverse ensemble [23].

2.– Different initialization points. In some learning algorithm that involve optimization of a non convex problem, the starting point in the algorithm can affect the solution. For example, altering the initial weights in a neural network can result in different predictors [24].

In the beginning of this section we mentioned that the output of an ensemble is obtained by combining the predictions of its base learners. The method in which this this combination is made is relevant to the performance of the ensemble. The most common combination techniques can be classified in two categories:

1.– Voting strategies. In this approach each base learner prediction is treated as a vote. In majority voting, the ensemble predicts the class label with the most votes. In weighted majority voting, each classifier is given a weight that determines the impact of its prediction in the final output, the ensemble chooses the label with the highest weighted tally.

2.– Non voting strategies. For some models that output a probability for each class, like logistic regressors, it is possible to use other combination strategies. In these cases the ensemble can use operators such as the minimum, maximum or mean of the confidences given by the base learners to determine the most probable class [25].

In addition to these possibilities there is another strategy known as stacking [26]. In contrast to the previously presented techniques stacking is a dynamic strategy, meaning the combination algorithm is not defined beforehand. In stacking, a new classifier is trained to take as feature vectors the output of the base learners and produce the output of the ensemble. This classifier can be seen as a meta learner stacked on top of the others.

Now we will describe in detail the strategies most relevant to our work: Bagging and subbagging.

2.3.1 Bagging and Subbagging.

Bagging [4] is an ensemble method in which the base predictors of the ensemble are built on bootstrap samples from the original train set. These bootstrap samples are built by drawing instances uniformly at random with replacement. Usually for standard bagging the bootstrap samples contain the same number of instances as the original, but variations can be made by adjusting their size.

It is important to notice that drawing with replacement will result in repeated instances. In fact approximately 63% of the instances will be unique and the rest repeated. This estimation is derived from the probability of an instance not being repeated in a sample size N as N increases. An instance will be unique in all other $N-1$ instances drawn are different. With uniform sampling this will occur with probability $1-1/N$ for each draw. As all draws are independent we get a total probability of

$$\left(1-\frac{1}{N}\right)^{N-1}, \tag{2.32}$$

applying a limit we get

$$\lim_{N\to\infty}\left(1-\frac{1}{N}\right)^{N-1}=\frac{1}{e}\simeq 0.63. \tag{2.33}$$

Training each base classifier in a different bootstrapped set aims to introduce diversity in the ensemble. However, for stable learners such as SVMs this technique alone may not be enough achieve sufficient variety.

The output of the ensemble is decided by majority voting. The pseudocode for bagging is detailed in algorithm (2.1).

Input: $\mathcal{D}_{train}$ % Train set
M % Ensemble size
$\mathcal{L}$ % Base learner
Output: E % Ensemble function
1 **for** $m \leftarrow 1$ ***to*** M **do**
2 $\quad \mathcal{D}_m \leftarrow Bootstrap(\mathcal{D}_{train}, replacement = True)$
3 $\quad c_m \leftarrow \mathcal{L}(\mathcal{D}_m)$
4 $E(\cdot) = \arg\max_{y \in \mathcal{Y}} \sum_{m=1}^{M} \mathbb{1}[c_m(\cdot) = y]$
5 **return** E

Algorithm 2.1: Bagging algorithm.

Subbagging [19] is a variation of the bagging algorithm in which instances are drawn without replacement. This has several implications. Bootstrap sets now will not have repeated instances, and therefore in order to be different their size must be smaller than that of the original set. If the original set has N_{train} instances a common choice for the bootstrap set size is $N_{train}/2$. This configuration has proven to be statistically equivalent to bagging. [27, 28]. As in bagging, the decision of the ensemble is determined by majority voting.

In both bagging and subbagging we have two opposing effects: The base learners trained on bootstrap samples will generally perform worse than a learner trained in the original train set because they are trained with fewer different instances. On the other hand, aggregating predictors decreases error by reducing the variance. As a consequence of these effects, bootstrap strategies are only effective when the variance reduction dominates the error increase derived from resampling [4].

Subbagging offers the advantage of reducing the complexity of training as a consequence of using smaller training sizes. Reducing the training time requirements for SVMs is precisely one of the main motivations of this work. For this reason we will explore this technique and some of its variations in homogeneous ensembles of SVMs in the next chapter, with the objective of achieving this goal without sacrificing the accuracy and robustness of a SVM.

3

Design development and implementation

In the previous chapter we have given a general overview of the classification problem in machine learning and of two populars models used for that task: SVM and ensemble methods. The goal of this work is to combine these two models in a way that takes advantage of their strengths and mitigates their drawbacks. During this project we have tried multiple ideas and designs to achieve this goal. In this chapter we will start by presenting, in section 3.1, the final design for the model along with two of its most relevant variations. Then, in section 3.2 we will the describe the key tools and methodology used for the development and implementation of the project.

3.1 Design

There are two key objectives towards which we have oriented our design. Firstly, the developed model needed to be at least as accurate as a single SVM, and secondly, its training computational complexity had to be low enough to be fit for problems with large volumes of data. The result of this design is a homogeneous ensemble with RBF kernel SVM as base classifiers. The RBF (Radial Basis Function) kernel is

$$k(\mathbf{x}, \mathbf{x}') = -\gamma||\mathbf{x} - \mathbf{x}'||^2. \tag{3.1}$$

The choice of this kernel in particular was motivated by the good results that have been achieved with it [29] and because it introduces a new parameter γ that will allow us to introduce more diversity in our classifiers when building the ensemble. This parameter controls the width of the kernel $(1/\gamma)$. Intuitively it can be thought of as the inverse of the influence radius of each support vector. With low values of γ all support vectors influence the classification of new instances, whereas with large values of γ only the closest support vectors to the new instance influence its classification.

As for the building of the ensemble, two techniques have been used to diversify the base SVMs and thus achieve an effective ensemble: Hyperparameter variation and subsampling. Subsampling in particular not only serves this purpose, it is also used as a means to reduce the computational

complexity of the model, which is one of our main goals. We will explain in detail how we applied this techniques later in this chapter. First we will outline the basic structure of the model. In order to do this we have to introduce first its two key parameters B and tp:

- **B** The ensemble is formed by B subensembles $\{E_b\}$. This subensembles will be comprised of SVMs that share the same hyperparameters (C, γ). Therefore the SVMs within the same subensemble only differ in the data they are trained on. The optimum value for B is problem dependent, nevertheless $B = 10$ is a good option in most cases.
- **tp**. (Short for *Train Proportion*) It determines the number of samples that will be used to train each base SVM in the ensemble in relation to the number of samples available for training. Having a train set $\mathcal{D}_{train}$ with N_{train} samples, each individual SVM will be trained with $tp \cdot N_{train}$ samples. As it is a proportion, we have $tp \in (0, 1)$.

One of the key characteristics of the model is that the individual SVMs within a subensemble E_b are trained in disjoint partitions of the train set. As a consequence, the parameter tp not only determines the size of the train samples for each base SVM but also determines the size of each subensemble, which is $1/tp$. Lower values of tp will then generate larger ensembles with SVMs trained with fewer instances, whereas high values of tp will generate smaller ensembles, but with SVMs trained with larger samples. In this way tp provides a trade-off between the strength of the individual classifiers and the size of the ensemble. In section 4.2 we will study how changing this parameter affects training complexity and accuracy of the classifier. Notice that, in conjunction, these parameters determine the size of the whole ensemble. With B subensembles each of size $1/tp$ we get a total ensemble size of B/tp.

The motivation for training the base SVMs within each subensemble in disjoint samples of the train set is that it increases their diversity, and thus improves the benefits of aggregation. This decision would not be feasible in most cases as it would greatly limit the number of base classifiers or the size of the train samples. However we are aiming our design at problems where a large volume of data is available for training and therefore, this is an acceptable limitation.

Having introduced the structure of the model we can now present in detail its training algorithm. We will break down this algorithm in two phases: SVM hyperparameter selection and individual SVMs training.

SVM hyperparameter tuning

As we saw previously, there are two key hyperparameters (C, γ) in SVMs with RBF kernel that need to be carefully adjusted in order to achieve a good performance [29]. In our model we will have B subensembles with common hyperparameters, so we will need to generate $\{C, \gamma\}_{b=1}^{B}$ B pairs of hyperparameters, one for each subensemble. Each pair $(C, \gamma)_b$ is obtained by performing an exhaustive grid search with 2-fold cross-validation over an independent partition of the train set $\mathcal{D}_{train,b}$. The pseudocode for the grid search is shown in Algorithm 3.1. We used the the grid $C = 2^q$, $\gamma = 2^q$ with $q = -5, -3, \ldots, 15$; $p = -15, -13, \ldots, 3$ proposed in [30]. We set the partitions $\mathcal{D}_{train,b}$ to have $2 \cdot N \cdot tp$ instances. In this way, during the grid search each SVM will be trained on $N \cdot tp$ instances, the number of instances that

each base SVM will be trained on later.

Having set the size of partitions $\mathcal{D}_{train,b}$ we distinguish two possible scenarios: If $tp \cdot B \leq 1/2$ then it is possible to make a partition of $\mathcal{D}_{train}$ with B disjoints subsets of $2 \cdot N \cdot tp$ instances each. If we have $tp \cdot B > 1/2$ then it is not possible to make such partition. In this case we take B subsets of $2 \cdot N \cdot tp$ instances drawn from $\mathcal{D}_{train}$ uniformly at random without replacement.

Individual SVM training

Once the hyperparameter pairs $\{C, \gamma\}_{b=1}^{B}$ have been selected, the individual SVMs are trained as follows: For each pair $(C, \gamma)_b$ a new random partition $\{\mathcal{D}_{train,b}\}_{t=1}^{T}$, with $T = 1/tp$ of the training set is generated. Then a SVM with hyperparameters $(C, \gamma)_b$ is trained in each subset $\mathcal{D}_{train,b,t}$. The result are $B \cdot T$ SVMs that form the final ensemble. The complete training pseudo-code is detailed in algorithm (3.2).

Input: C % List of values for parameter C
γ % List of values for parameter γ
$\mathcal{D}$ % Training set
n % Number for folds in cross-validation
Output: $(\tilde{C}, \tilde{\gamma})$ % Best combination of values for parameters C and γ

$N \leftarrow size(\mathcal{D})$
$N_{val} \leftarrow \frac{1}{n} \cdot N$
$N_{train} \leftarrow N - N_{val}$
for $i \leftarrow 1$ ***to*** $len(C)$ **do**
  for $j \leftarrow 1$ ***to*** $len(\gamma)$ **do**
    for $k \leftarrow 1$ ***to*** n **do**
      $\mathcal{D}_{train}, \mathcal{D}_{val} \leftarrow Split(\mathcal{D}, N_{train}, N_{val})$
      $c^{(k)} \leftarrow SVM_{\{C^{(i)}, \gamma^{(j)}\}}(\mathcal{D}_{train})$
      $score^{(i,j,k)} \leftarrow evaluate(^{(k)}, \mathcal{D}_{val})$
$meanScores^{(i,j)} \leftarrow mean(score^{(i,j,k)}, k)$
$\tilde{i}, \tilde{j} \leftarrow \arg\max_{i,j} \left(meanScores^{(i,j)}\right)$
$\tilde{C} \leftarrow C^{\tilde{i}}$
$\tilde{\gamma} \leftarrow \gamma^{\tilde{j}}$
return $\tilde{C}, \tilde{\gamma}$

Algorithm 3.1: Exhaustive grid search with n fold cross-validation.

For the classification of a new instance **x**, the model follows a standard majority voting strategy. Each SVM $\{c_m\}_{m=1}^{M}$ in the ensemble E, makes a prediction $c_m(\mathbf{x})$. The class chosen by the ensemble is

$$E(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{m=1}^{M} \mathbb{1}[c_m(\mathbf{x}) = y] \tag{3.2}$$

Input: $\mathcal{D}_{train} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{train}}$ % Training set
B % Number of subensembles
tp % Train proportion

Output: E % Ensemble

if $tp \leq 1/2B$ **then**
  $\{\mathcal{D}_i\}_{i=1}^{B} \leftarrow$ Split $\mathcal{D}_{train}$ in B disjoint samples of size $2 \cdot tp \cdot N_{train}$
else
  $\{\mathcal{D}_i\}_{i=1}^{B} \leftarrow$ Split $\mathcal{D}_{train}$ in B samples of size $2 \cdot tp \cdot N_{train}$ drawn uniformly at random.
$T = 1/tp$
foreach $b \leftarrow 1$ ***to*** B **do**
  $\{C, \gamma\}_b \leftarrow GridSearch(\mathcal{D}_b, folds = 2)$
  $\{\mathcal{D}_t\}_{t=1}^{T} \leftarrow$ Split $\mathcal{D}_{train}$ in T disjoint subsets of size $tp \cdot N_{train}$
  foreach $t \leftarrow 1$ ***to*** T **do**
    % Train a new SVM with hyperparameters $\{C, \gamma\}_b$ and data $\mathcal{D}_t$
    $c_{b,t} \leftarrow SVM_{\{C,\gamma\}_b}(\mathcal{D}_t)$
$E(\cdot) = \arg\max_{y \in \mathcal{Y}} \sum_{b=1,t=1}^{B,T} \mathbb{1}\left[c_{b,t}(\cdot) = y\right]$
return E

Algorithm 3.2: SVM ensemble training.

3.1.1 Model variations

We will now present some ot the variations explored during the design of the model. Some of these variations proved to be useful in some scenarios and can be used by adjusting some parameters of the model, others were not and were simply discarded.

Elimination of bad hyperparameters

During the first part of the training we obtain B pairs of hyperparameters $\{C, \gamma\}_{b=1}^{B}$. Because of the way in which these are selected these pairs are not necessarily different, in fact repetition of some of them is common, specially for high values of tp. We evaluated the impact that the diversity of these pairs had on the performance of the model. In order to do this we studied the influence of each individual subensemble $\{E_b\}_{b=1}^{B}$ on the accuracy of the whole ensemble. By doing so we observed that some of them produced a noticeable increase in training error. For this reason,we considered adding an extra step in the training algorithm in which the subensembles $\{E\}_{b=1}^{B}$ were evaluated and removed if they performed below a certain threshold. However, this attempt was unsuccessful. The elimination of this classifiers often lowered the training error but it did not improve accuracy in test.

Extra classifiers

There is an extra parameter for the model called $extra\_machines$. This parameter is used to increase the size of the ensemble without changing B or tp. Increasing this parameter is particularly useful in scenarios where a low B is chosen, as it results in a small ensemble size that may not have converged yet. With this parameter the ensemble size is given by the formula: $B \cdot tp \cdot (1 + extra\_machines)$.

The default values for $extra\_machines$ is 0. With this value the ensemble is built as described previously. For a different value, $extra\_machines = K$, the training algorithm consists of K repetitions of the standard training procedure resulting in $\{E_k\}_{k=1}^{K}$ ensembles. This ensembles are simply added together to construct the final ensemble E.

3.2 Development and implementation

The model design we presented in the previous section was not fixed at the beginning of the project. It was rather the result of a research iterative process in which small changes were studied and gradually introduced into the algorithm until a satisfactory result was achieved.

We began the project by defining our goal and framing our problem. After a brief analysis phase we developed a simple initial version of our model. Once this was done, the iterative phase began. Each iteration started with a meeting in which we first analyzed the results and progress made during the previous iteration, and then, we proposed variations that could potentially improve the model. These variations were then introduced and tested. If the results of these tests were positive then the modifications were kept, if not they were discarded. To keep track of the different versions of the model we used the version control system Git. This tool was also used to control de different versions of the tests generated during all the development of the project.

The project was implemented using Python and R. The classifier was entirely built in Python using the machine learning library scikit-learn [31]. The tests were primarily implemented in Python, and R was used for some tasks such as the generation of synthetic datasets and the realization of standard statistical tests. Processing and visualization of the results obtained in the tests was done in Python using the libraries Pandas, NumPy and Matplotlib.

4

Tests and results

In the previous chapter we presented in detail the model developed in this project. The motivation behind its design was to produce a model with low train complexity so it could be trained in large datasets where using a SVM is unfeasible. However, this improvement in training time should be achieved without sacrificing the accuracy and robustness of the SVM. In this chapter we present the tests carried out to evaluate several aspects of the developed model. In section 4.1 we study the training and predictions time of our model in comparison with those of the SVM. Then, in section 4.2 we evaluate the accuracy of the proposed with respect to the SVM. Finally, in section 4.3 we compare our model to an ensemble of similar characteristics built with subbagging.

These tests have been carried out in 3 synthetic datasets and 3 real datasets. The synthetic datasets [32] used are: *Twonorm*, *Threenorm* and *Ringnorm*. The real datasets are taken form the UCI repository [33] and are: *Magic04*, *Bank Marketing* [34] and *Adult*. The characteristics of these datasets are detailed in Table 4.1.

Dataset	Instances	Attributes
Magic04	19020	14
Bank	45211	17
Adult	48842	14
Twonorm	synthetic	20
Threenorm	synthetic	20
Ringnorm	synthetic	20

Table 4.1: Datasets.

Before executing any test some basic preprocessing has been applied to the data. Both Adult and Bank datasets have some categorical attributes. These attributes were transformed into numerical features using one hot encoding. Then, all features from all datasets were standardized by removing the mean and scaling to unit variance.

All time measurements in the tests we carried out have been taken from single-threaded executions on an Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz with 8 GB RAM. The tests that did not require time

execution measurements were executed in the Centro de Computación Científica (CCC) at UAM.

4.1 Training and prediction complexity

In this section we will study the training and prediction complexity of the developed model. For both training and prediction we will first make estimations from a theoretical point of view and then we will compare them to the results obtained with an empirical evaluation. In order to put the results of our model in perspective, we will compare them to those of a single SVM.

4.1.1 Training

We will express the training complexity of our model as a function of 4 factors: N_{train}, $GSize$, B and tp, where N_{train} is number of samples in the available training set, (B, tp) are hyperparameters of the model (see Section 3.1) and $GSize$ is the the size of the grid over which the hyperparameters (C, γ) are selected. The pseudocode for the training is detailed in Algorithm 3.2 in Chapter 3. Breaking down the training algorithm in its phases we get:

1.– Hyperparameters selection. An exhaustive 2 fold cross-validation search is performed over the grid with train subsets of size $tp \cdot N_{train}$. This process is repeated B times. Assuming a train complexity of $O(N^2)$ for a SVM on N samples and ignoring the evaluation cost we get the following cost:

$$B \cdot 2 \cdot GSize \cdot O(tp^2 \cdot N_{train}^2). \tag{4.1}$$

2.– Base SVMs training. The ensemble is formed by $B \cdot 1/tp$ SVMs trained on $tp \cdot N_{train}$ samples. The resulting cost is:

$$B \cdot \frac{1}{tp} \cdot O(tp^2 \cdot N_{train}^2). \tag{4.2}$$

Adding up these costs we get an estimate of the overall complexity:

$$B \cdot 2 \cdot GSize \cdot O(tp^2 \cdot N_{train}^2) + B\frac{1}{tp} \cdot O(tp^2 \cdot N_{train}^2) = B \cdot O(tp^2 \cdot N_{train}^2)(2 \cdot GSize + \frac{1}{tp}). \tag{4.3}$$

In most situations , we will have $2 \cdot GSize > 1/tp$. This implies that the search of hyperparameters phase is more demanding than the training of the base learners. The opposite would be the case if a very low value for tp was used, (which might be adequate for very large datasets) or if the hyperparameter grid was explored with alternative techniques like randomization or partial optimization [30] that lower the effective size of the grid.

The grid used in this work has $GSize = 110$ and a the values explored for tp are higher than 0.005

so we will eliminate the base learners training term from the equation (4.3) and arrive to the following expression for the training complexity:

$$O(GSize \cdot B \cdot tp^2 \cdot N_{train}^2). \tag{4.4}$$

To put this results in perspective we estimate the cost of training a single SVM performing a grid search to tune its hyperparameters. We will assume an exhaustive grid search with n fold cross-validation. In this estimation we ignore the evaluation cost of the grid search. The pseudocode of the training algorithm with $n = 10$ is presented in Algorithm (4.1).

Input: $\mathcal{D}_{train}$ % Training set
Output: c % Trained classifier
1 $Grid = \{C, \gamma\} : C = 2^q, \gamma = 2^q$ with $q = -5, -3, \ldots, 15$; $p = -1, 1, \ldots, 13$
2 $\{C, \gamma\} \leftarrow GridSearch(Grid, \mathcal{D}_{train}, cv = 10)$
3 $c \leftarrow SVM(\{C, \gamma\}, (\mathcal{D}_{train})$
4 **return** c

Algorithm 4.1: Single SVM training pseudocode.

The cost of the grid search is:

$$GSize \cdot n \cdot O\Big(\Big(\frac{n-1}{n}\Big)^2 N_{train}^2\Big), \tag{4.5}$$

and the cost of tuning the SVM itself is:

$$O\Big(N_{train}^2\Big). \tag{4.6}$$

In this case is clear that the hyperparameter tuning is the dominant term in the overall cost. Therefore we can write the estimation for the training cost of the single SVM as:

$$O\Big(GSize \cdot \frac{(n-1)^2}{n} \cdot N_{train}^2\Big). \tag{4.7}$$

From the expressions (4.4) and (4.7) we observe that both models train complexity scales quadratically with the number of instances on the train set. The ratio between them is

$$O\Big(\frac{(n-1)^2}{n \cdot B \cdot tp^2}\Big). \tag{4.8}$$

This indicates that even though their training complexity have an equivalent asymptotic growth, the ensemble training cost can be adjusted using its parameter B and tp to achieve significant speedups. In the next section 4.2 we will study how changing this parameters affects the performance of the

ensemble. From equation 4.4 we observe that the training complexity for the ensemble scales linearly with B and quadratically with tp. We will now present the results of an empirical evaluation to see how accurate are the complexity estimations presented up to this point.

Results

In these tests we study the training time for the ensemble and for the single SVM as a function of the size of the train set N_{train} and in function of tp. The tests have been carried out for two datasets: Magic and Twonorm. The reported results are averages over 10 independent executions.

In the first test we study the impact of N_{train}. For the ensemble we have fixed $B = 10$ $tp = 0.2$. For the SVM we used 10 fold cross-validation in the grid search, $n = 10$ Both models use the same grid with $GSize = 110$. The models were evaluated for N_{train} = [100, 200, 400, 800, 1600, 3200]

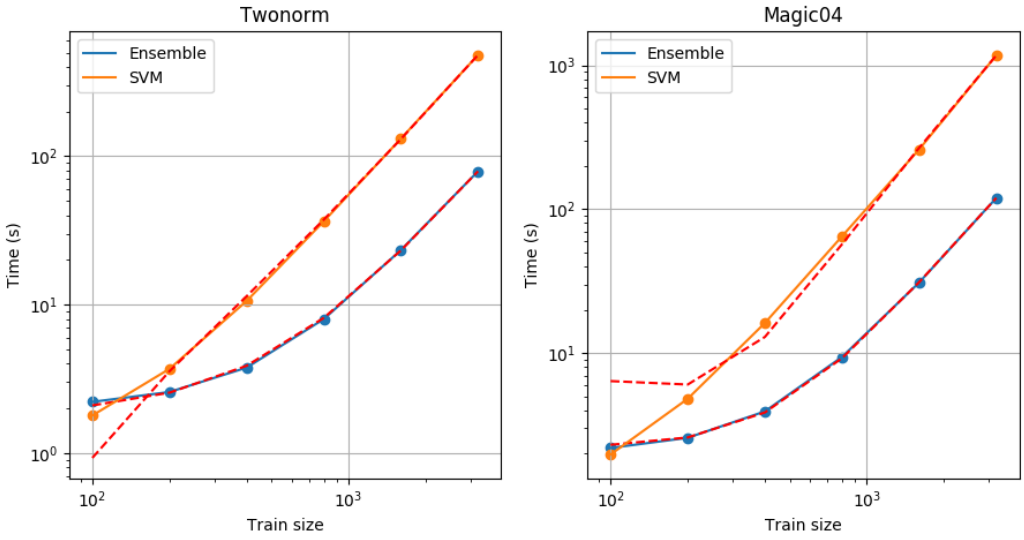

Figure 4.1: Training Time for Ensemble and SVM in datasets Twonorm (left) and Magic (right). Both axis are presented with a logarithmic scale. The red dashed line indicates a quadratic fit to the data.

The results are presented in Figure 4.1 in logarithmic scale in both axis to better appreciate the growth of the functions. A quadratic polynomial was fitted to the data and is represented by the red dashed line. These graphs suggest that our estimations are correct and that both models training time experience a quadratic increase with the size of the training set.

In the next test we study the training time for the ensemble as a function of its hyperparameter tp. The parameter B and the training set size N_{train} are fixed with values $B = 10$, $N_{train} = 10000$. The ensemble is evaluated for tp = [0.005, 0.01, 0.02, 0.25, 0.04, 0.05, 0.0625, 0.1].

As we did for the previous test, the results are presented Figure 4.2 with a quadratic fit to the data marked with a red dashed line. We observe again that the data clearly follow a quadratic trend, so we can confirm the cost estimation we obtained earlier in this section. It is also worth noting that there is

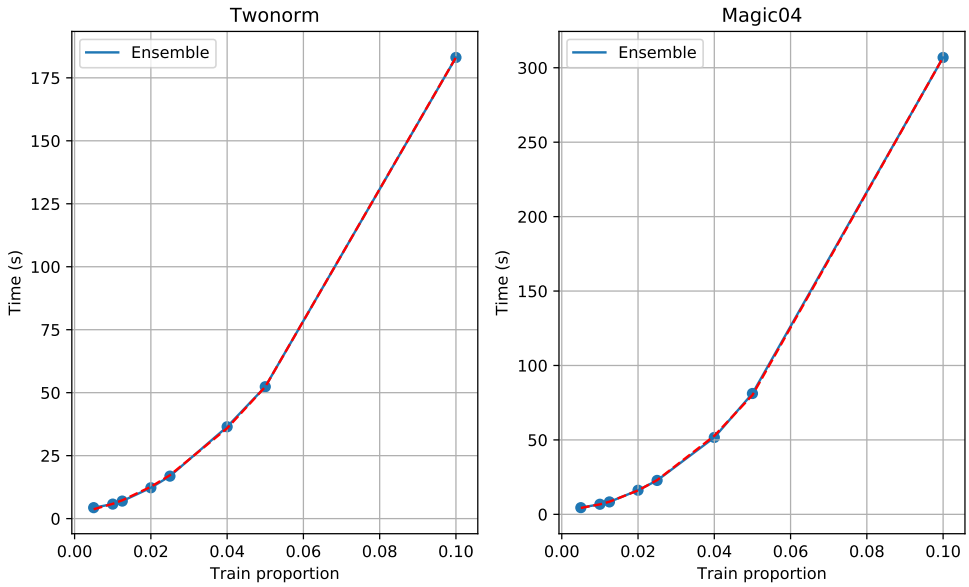

Figure 4.2: Training Time for Ensemble in datasets Twonorm (left) and Magic (right). The red dashed line indicates a quadratic fit to the data.

a substantial difference between both problems. Training in Magic is approximately twice as slow as in Twonorm for the SVM and 1.5 times as slow for the ensemble. This shows that training time for both models is very problem dependent.

4.1.2 Prediction

As we saw in section 2.2.1 classifying a new instance with a SVM requires a kernel evaluation for each support vector. The kernel evaluation is the most computationally expensive operation of the process. Therefore, the prediction cost increases linearly with the number of the support vectors. Unfortunately, the number of support vectors of a SVM cannot be determined with precision before it is trained. As a consequence, the best method to determine prediction cost is to estimate it with empirical evaluation.

Despite not being able to determine accurately the number of support vectors of an SVM beforehand its relevant to mention two general notions that can give us a vague idea of how this number may change.

1.– There is an inverse dependence between the number of support vectors and the regularization parameter C. This is a consequence of the flexibility that C allows in the model. With low values of C more instances are allowed to be misclassified or to lie inside the margin, which become support vectors. By contrast, a high value of C results in hard margins, with less misclassified instances and thus, less support vectors

2.– As general rule we can expect the number of support vectors to increase with the size training set. This rule is justified with a probabilistic argument: With more samples used in training there is a bigger chance of having data points near the decision boundary that become support vectors.

In an ensemble, classification of a new instance requires the classification of each base learner

and the combination of their outputs. Therefore for an homogeneous ensemble the cost of prediction increases linearly with its size.

In our model, the combination of outputs consists of a simply vote tally and therefore has a negligible impact on the overall prediction cost. The prediction cost is then determined by the size of the ensemble and the classification cost for each base learner. Determining the impact of parameter B is straightforward. Increasing its value increases linearly the size of the ensemble and thus, the prediction cost. The effect of the tp is however harder to determine as it creates two opposing effects. On the one hand, increasing tp generates base SVMs trained with more instances and potentially slower in their predictions. On the other hand, increasing tp reduces the ensemble size and therefore, the overall prediction cost.

We present now the results of the experiments carried out and the conclusions drawn from them.

Results

In these tests we study the prediction time for the ensemble and the SVM as a function of the size of the train set N_{train} and of the parameter tp. The tests have been carried out in two datasets: Magic and Twonorm, measuring the time needed for classification of 1000 instances. The reported results are averages over 10 independent executions.

In the first test we study the impact of N_{train}. Both models have been trained with the same parameters as the previous test. For the ensemble this is $B = 10$, $tp = 0.2$ and for the SVM $n = 10$. The models were evaluated for N_{train} = [100, 200, 400, 800, 1600, 3200].

In figure 4.3 we observe that classification cost also seems to have a quadratic dependence with the size of the training set for both models. We also observe that the ensemble has lower classification times than the SVM in all cases with the exception of $N_{train} = 100$ in Twonorm. It is important to notice that here too there is a substantial difference between the classification times of both problems. For Twonorm the classification is up to 5 times faster than for Magic using the ensemble and up to 12 times faster using the SVM.

In the second test we study the prediction time for the ensemble in function of tp. The parameter B and the training set size N_{train} are fixed with values $B = 10$, $N_{train} = 10000$. The ensemble is evaluated for tp = [0.005, 0.01, 0.02, 0.025, 0.04, 0.05, 0.0625, 0.1] which correspond to ensemble sizes of [2000, 1000, 500, 400, 250, 200, 160, 100] SVMs.

From figure 4.4 we clearly see that an increase of tp results in lower classification cost. This indicates that the size of the ensemble is more relevant than the prediction cost of the individual SVMs.

The results obtained indicate that the SVM classification cost is largely problem dependent and its difficult to estimate beforehand. For our ensemble this implies that the only factors we should rely on to estimate classification cost are training size and ensemble size.

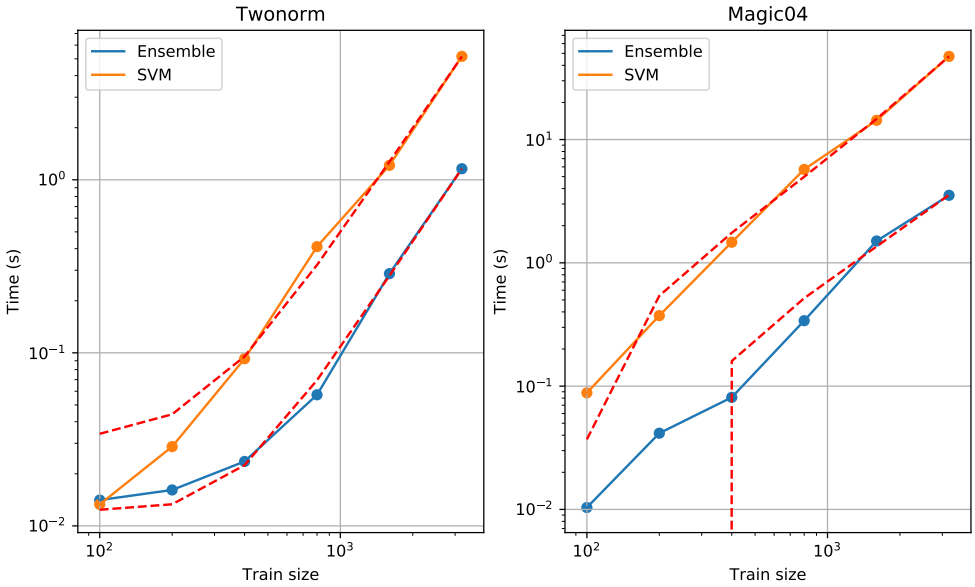

Figure 4.3: Times for classification of 1000 instances with Ensemble and SVM in datasets Twonorm (left) and Magic (right). Both axis are presented with a logarithmic scale. The red dashed line represents a quadratic fit to the data.

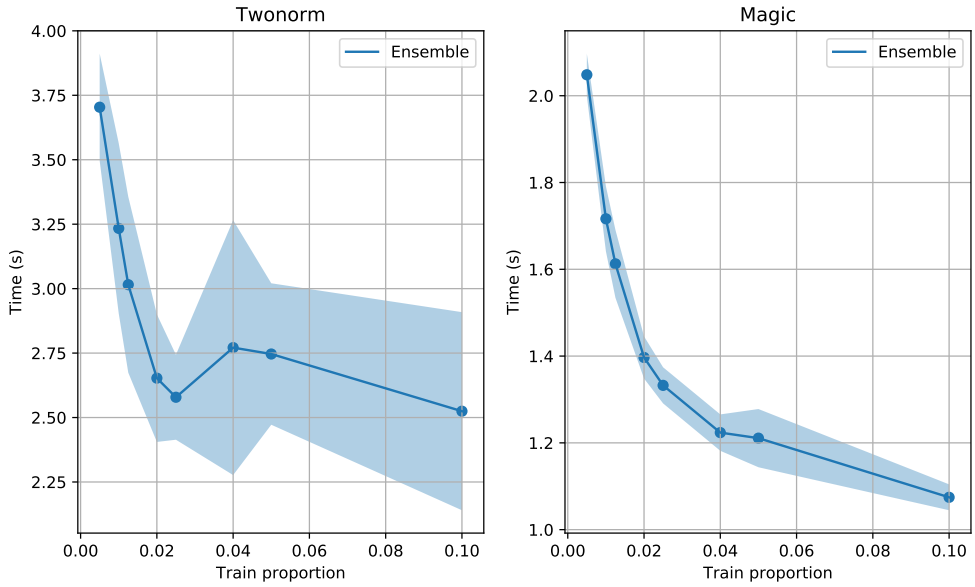

Figure 4.4: Times for classification of 1000 instances with the ensemble in datasets Twonorm (left) and Magic (right). The shaded region represents the standard deviation of the time measurements.

4.2 Accuracy comparison between Ensemble and SVM

As we presented in Section 2.2, SVMs are reliable models for classification tasks. However, their training cost quickly becomes prohibitive as the size of the training set increases. The training requires solving a quadratic programming problem which, in practice, has a complexity $O(N_{train}^2)$ for a train set of N_{train} instances. Moreover, RBF kernel SVMs in particular, heavily rely in the proper tuning of hyperparameters $\{C, \gamma\}$ to achieve good performance [29]. This tuning is usually performed using a grid search with cross-validation. This is a computationally intensive process that aggravates the previous problem.

With this limitations in mind, we designed this test to compare the performance of a single SVM and the ensemble in scenarios where training time is constrained. For each problem we fixed a maximum amount of available training instances (10000) for both the SVM and the ensemble. The use of this data however, was limited by the established time constraints.

For a single SVM the most natural way of adjusting to these constraints is to reduce the number of instances used for training. In this way, the SVM will use only a fraction of the available training set.

For the ensemble, the parameters B and tp can be reduced to adjust to the constraints while using all the available data. In these tests we only modify tp and leave B fixed at $B = 10$.

4.2.1 Methodology

The two models compared in this test are a single SVM and the ensemble developed in this project. For the single SVM we fixed a list of train sizes (≤ 10000) and for the ensemble a list of values for the hyperparameter tp. Each model accuracy and train time were then evaluated for each of these values in order to compare both models performances given the same amount of training time.

The single SVM model was trained in two steps: First, the hyperparameters $\{C, \gamma\}$ are tuned with an exhaustive grid search with 10 fold cross-validation. The grid of values is the same as the one described in Section 3.1 ($C = 2^q$, $\gamma = 2^q$ with $q = -5, -3, \ldots, 15$; $p = -1, 1, \ldots, 13$). Finally, the SVM with the chosen hyperparameters is trained on the complete training set. This model training pseudocode is detailed in Algorithm 4.1. The ensemble is trained with the algorithm described in Algorithm 3.2.

The test for synthetic datasets and real datasets are slightly different due to the limited amount of data available of the latter. They are described in detail for the single SVM and the ensemble in the Appendix A.

4.2.2 Results

In this section we present the results of the test previously described. For the N_{train} Single SVM model the training set sizes evaluated are: [200, 500, 1000, 2000, 3500, 5000, 10000] for all problems except Adult, for which the last value (10000) was not evaluated due to the computational limitations of our workstation. For the ensemble model the tp evaluated are: [0.01, 0.025, 0.05, 0.1, 0.2, 0.25]. The scores presented are the result of averaging 20 independent executions of the test, the training time measurements are the result of averaging 4 executions. This difference in the number of repetitions is due to two reasons: Time measurements are more stable and therefore increasing the number of repetitions does not improve the precision of the measurement. Secondly, they are harder to make because they must be taken in a controlled environment with a single thread execution unlike the scores that can be obtained with parallel executions on different machines or in a computation cluster.

The Figures 4.5 , 4.6 represent the scores obtained by each model in the vertical axis with their corresponding training time in the horizontal axis, represented in logarithmic scale. The standard deviation of the score is represented by the shaded region. These results are also presented in Tables 4.2 and 4.3.

From Figures 4.5 , 4.6 we observe that in 4 of the 6 problems the accuracy of the ensemble improves when the value of tp increases. In the other 2, Twonorm and Ringnorm the maximum accuracy was reached for low values of tp and remained equal for the higher values. This trend suggest that in general, the improvement on the accuracy of the base SVMs was more relevant that the benefits obtained by having a larger ensemble size. For tp values of 0.01 and 0.025 we had ensembles of sizes 1000 and 400 respectively. We could argue that the ensemble predictions converge before reaching these sizes, and therefore it does not benefit from having more classifiers.

Comparing both models we observe that for a fixed amount of training time the ensemble consistently outperforms the single SVM. The difference between their accuracies is specially significant for the cases with low training time ($< 100s$).

Comparing maximum scores, regardless of training time we observe that the ensemble has the highest score in 3 datasets: Adult, Twonorm and Ringnom, while the SVM has the highest score in the other 3. However, the highest scores of the SVM are always achieved for $N_{train} = 10000$ at a high computational cost. Comparing the training times of the highest scores achieved by each model we observe that the SVM has training time increase factors of 7.6, 36.8 and 6.6 for Magic, Bank and Threenorm respectively (see Tables 4.2 4.3). In all the studied problems the difference between the maximum scores of both models is not very significant, always being less than $1,5$ times the standard deviation of any of the scores.

With this analysis we arrive at two conclusions: Firstly, both models have comparable accuracies when given an unlimited amount of training time, and secondly, the ensemble consistently achieves better scores than the SVM for any fixed amount of training time.

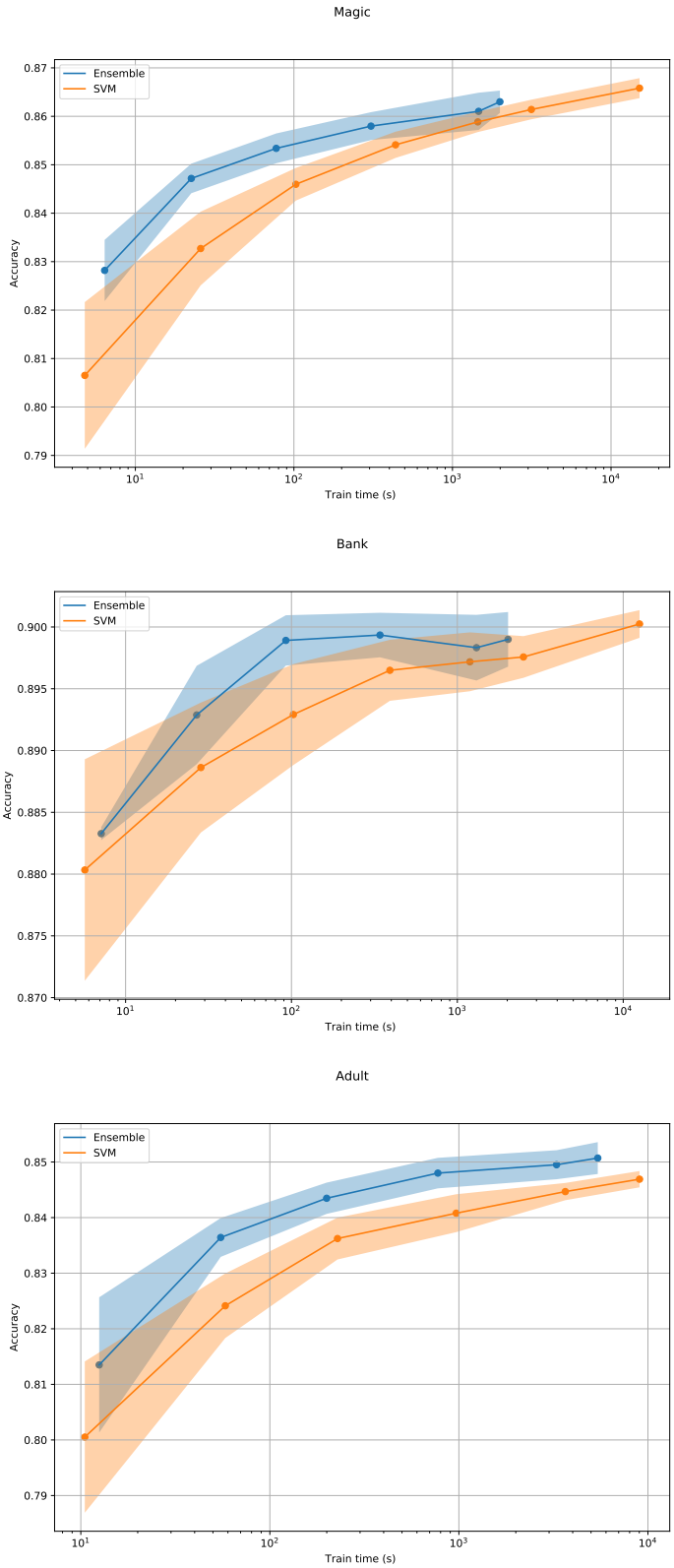

Figure 4.5: Accuracy comparison between Ensemble and SVM in the real datasets: Magic, Bank and Adult. The vertical axis represents the score of the models while the horizontal axis represent the training time in a logarithmic scale. The shaded region represents the standard deviation of the scores obtained.

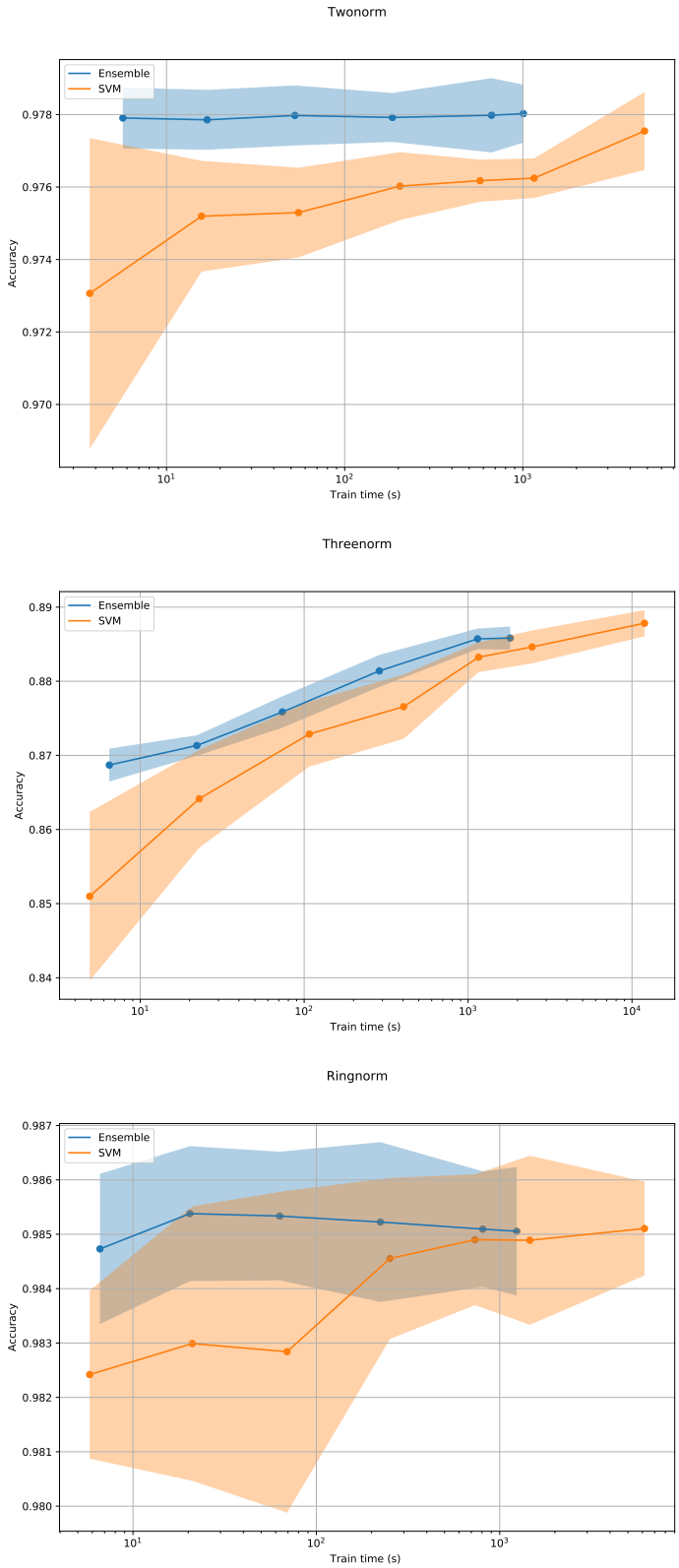

Figure 4.6: Accuracy comparison between Ensemble and SVM in the synthetic datasets: Twonorm, Threenorm and Ringnorm. The vertical axis represents the score of the models while the horizontal axis represent the training time in a logarithmic scale. The shaded region represents the standard deviation of the scores obtained.

SVM Ensemble		
tp	Score	Time (s)
0.010	82.82±0.63	6.40±0.20
0.025	84.72±0.31	22.54±1.28
0.050	85.34±0.31	77.34±0.47
0.100	85.80±0.29	306.35±3.05
0.200	86.10±0.38	1462.35±118.66
0.250	86.30±0.23	1991.58±13.51

(a) Magic. Ensemble.

Single SVM		
N_{train}	Score	Time (s)
200	80.65±1.51	4.80±0.37
500	83.27±0.76	25.81±2.44
1000	84.60±0.34	102.87±1.18
2000	85.41±0.27	437.49±57.67
3500	85.88±0.21	1445.34±53.12
5000	86.14±0.20	3145.13±129.61
10000	86.58±0.21	15127.35±514.01

(b) Magic. Single SVM.

SVM Ensemble		
tp	Score	Time (s)
0.010	88.33±0.05	7.13±0.03
0.025	89.29±0.40	26.84±0.20
0.050	89.89±0.20	92.52±0.33
0.100	89.93±0.18	341.74±1.84
0.200	89.83±0.27	1304.53±9.86
0.250	89.90±0.22	2023.25±30.18

(c) Bank. Ensemble.

Single SVM		
N_{train}	Score	Time (s)
200	88.03±0.90	5.68±0.11
500	88.86±0.53	28.44±1.08
1000	89.29±0.41	102.69±1.55
2000	89.65±0.25	392.08±7.23
3500	89.72±0.24	1190.67±19.23
5000	89.76±0.17	2503.75±70.17
10000	90.02±0.11	12543.61±64.57

(d) Bank. Single SVM.

SVM Ensemble		
tp	Score	Time (s)
0.010	81.35±1.22	12.51±0.10
0.025	83.64±0.35	54.97±0.25
0.050	84.35±0.28	199.76±0.58
0.100	84.80±0.27	774.11±6.40
0.200	84.95±0.26	3285.41±36.97
0.250	85.07±0.29	5428.58±45.47

(e) Adult. Ensemble.

Single SVM		
N_{train}	Score	Time (s)
200	80.05±1.36	10.50±0.10
500	82.41±0.58	58.12±1.46
1000	83.62±0.38	227.98±8.40
2000	84.08±0.34	967.83±34.18
3500	84.47±0.16	3654.76±137.48
5000	84.69±0.15	9027.30±833.47

(f) Adult. Single SVM.

Table 4.2: Average scores and training times for the real datasets.

SVM Ensemble		
tp	Score	Time (s)
0.010	97.79±0.08	5.69±0.02
0.025	97.79±0.08	16.91±0.07
0.050	97.80±0.08	52.51±0.12
0.100	97.79±0.07	185.27±0.31
0.200	97.80±0.10	667.53±4.54
0.250	97.80±0.08	1008.01±7.60

(a) Twonorm. Ensemble.

Single SVM		
N_{train}	Score	Time (s)
200	97.31±0.43	3.71±0.04
500	97.52±0.15	15.73±0.08
1000	97.53±0.12	55.07±0.82
2000	97.60±0.09	204.13±1.66
3500	97.62±0.06	575.05±10.08
5000	97.62±0.05	1156.25±10.82
10000	97.75±0.11	4819.12±37.73

(b) Twonorm. Synthetic.

SVM Ensemble		
tp	Score	Time (s)
0.010	86.87±0.22	6.47±0.04
0.025	87.13±0.14	22.15±0.07
0.050	87.59±0.21	73.63±0.97
0.100	88.14±0.22	287.44±3.79
0.200	88.57±0.14	1142.24±19.12
0.250	88.58±0.16	1806.76±16.54

(c) Threenorm. Ensemble.

Single SVM		
N_{train}	Score	Time (s)
200	85.10±1.14	4.92±0.06
500	86.41±0.66	22.90±0.78
1000	87.29±0.44	107.42±7.76
2000	87.66±0.43	403.86±34.77
3500	88.32±0.20	1159.19±17.81
5000	88.46±0.22	2456.30±48.17
10000	88.78±0.18	11923.65±273.75

(d) Threenorm. Single SVM.

SVM Ensemble		
tp	Score	Time (s)
0.010	98.47±0.14	6.60±0.03
0.025	98.54±0.12	20.44±0.06
0.050	98.53±0.12	63.19±0.27
0.100	98.52±0.15	223.34±2.53
0.200	98.51±0.11	808.90±3.21
0.250	98.51±0.12	1238.11±3.64

(e) Ringnorm. Ensemble

Single SVM		
N_{train}	Score	Time (s)
200	98.24±0.15	5.81±0.07
500	98.30±0.25	21.06±2.08
1000	98.28±0.30	69.25±0.59
2000	98.46±0.15	251.93±0.49
3500	98.49±0.12	733.09±8.12
5000	98.49±0.16	1459.79±8.94
10000	98.51±0.09	6163.33±75.59

(f) Ringnorm. Single SVM.

Table 4.3: Average scores and training times for the synthetic datasets.

4.3 Comparison to standard subbagging

As we presented in Section 2.3, for bagging and subbagging methods to be effective the reduction of variance in the error achieved by aggregation has to dominate over the error increase on the base learners due to bootstrap sampling.

In the model proposed in this work we focused on modifying the standard subbagging method by sampling with a certain structure instead of sampling uniformly at random. By training the base learners in disjoint subsets of the training set we hope to maximize diversity of the base classifiers, improving the aggregation benefits, while minimizing the loss of accuracy in base learners by using all the available data.

We designed this test to evaluate the effects of this structured sampling. The models compared are the ensemble developed in this work and an ensemble of similar characteristics built with standard subbagging instead. In this section we will refer to this models as *Structured* and *Subbagging* respectively.

Both models are built following the same algorithm, the only exception being the way in which the train sets for the base learners are drawn. Therefore, the Subbagging model also has hyperparameters B, tp analogous to those of Structured. The training pseudocode for Subbagging model is detailed in Algorithm 4.2 The pseudocode for the Structured model is described in Algorithm 3.2.

Input: $\mathcal{D}_{train} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{train}}$ % Training set
B % Number of batches
tp % Train proportion

Output: E % Ensemble

if $tp \leq 1/2B$ **then**
  $\{\mathcal{D}_i\}_{i=1}^{B} \leftarrow$ Split $\mathcal{D}_{train}$ in B disjoint subsets of size $2 \cdot tp \cdot N_{train}$
else
  $\{\mathcal{D}_i\}_{i=1}^{B} \leftarrow$ Split $\mathcal{D}_{train}$ in B subsets of size $2 \cdot tp \cdot N_{train}$ drawn uniformly at random.
$T = 1/tp$
foreach $b \leftarrow 1$ ***to*** B **do**
  $\{C, \gamma\}_b \leftarrow$ GridSearch($\mathcal{D}_b$)
  foreach $t \leftarrow 1$ ***to*** T **do**
    $\{\mathcal{D}_t\}_{t=1}^{T} \leftarrow Bootstrap(\mathcal{D}_{train}, size = tp \cdot N_{train}, replacement = False)$
    % Train a new SVM with hyperparameters $\{C, \gamma\}_b$ and data $\mathcal{D}_t$
    $c_{b,t} \leftarrow SVM_{\{C,\gamma\}_b}(\mathcal{D}_t)$
$E(\cdot) = \arg\max_{y \in \mathcal{Y}} \sum_{b=1,t=1}^{B,T} \mathbb{1}[c_{b,t}(\cdot) = y]$
return E

Algorithm 4.2: Subbagging ensemble training.

4.3.1 Results

For this test the ensembles were evaluated with different values for the hyperparameter tp: [0.01, 0.02, 0.025, 0.05, 0.1, 0.2]. The hyperparameter B remained fixed for both models at $B = 10$. The mean and standard deviations reported are the average over 20 independent executions of the test.

In the Figure 4.7 we present the results of the real datasets: Magic, Bank and Adult; and in Figure 4.8 those of the synthetic datasets: Twonorm, Threenorm and Ringnorm. In both figures the plots in the left column display the average score of both models in relation to their tp value. In this graph the shaded region represents the standard deviation of the scores, which is also represented in the graphs of the right column.

These results show that the average accuracy of both models is almost identical in all problems. However the structured model produces more stable result than regular subbagging. This can be deduced from the standard deviation graphs, where the curve for subbagging stays on top of the curve for the structured model. This difference is more noticeable for low values of tp and is particularly interesting in the synthetic problems. By observing the left plots of Figure 4.8 we see that in these problems the standard deviation for Structured is almost 0 and constant for all tp values. By contrast, the subbagging model shows several pronounced peaks in its graph. This suggests that although both models perform similarly on average, the random sampling of subbagging makes it less stable than its structured counterpart.

We can conclude that using the structured approach over regular subbagging does not yield better results on average but it produces more stable ensembles, specially for low sampling ratios.

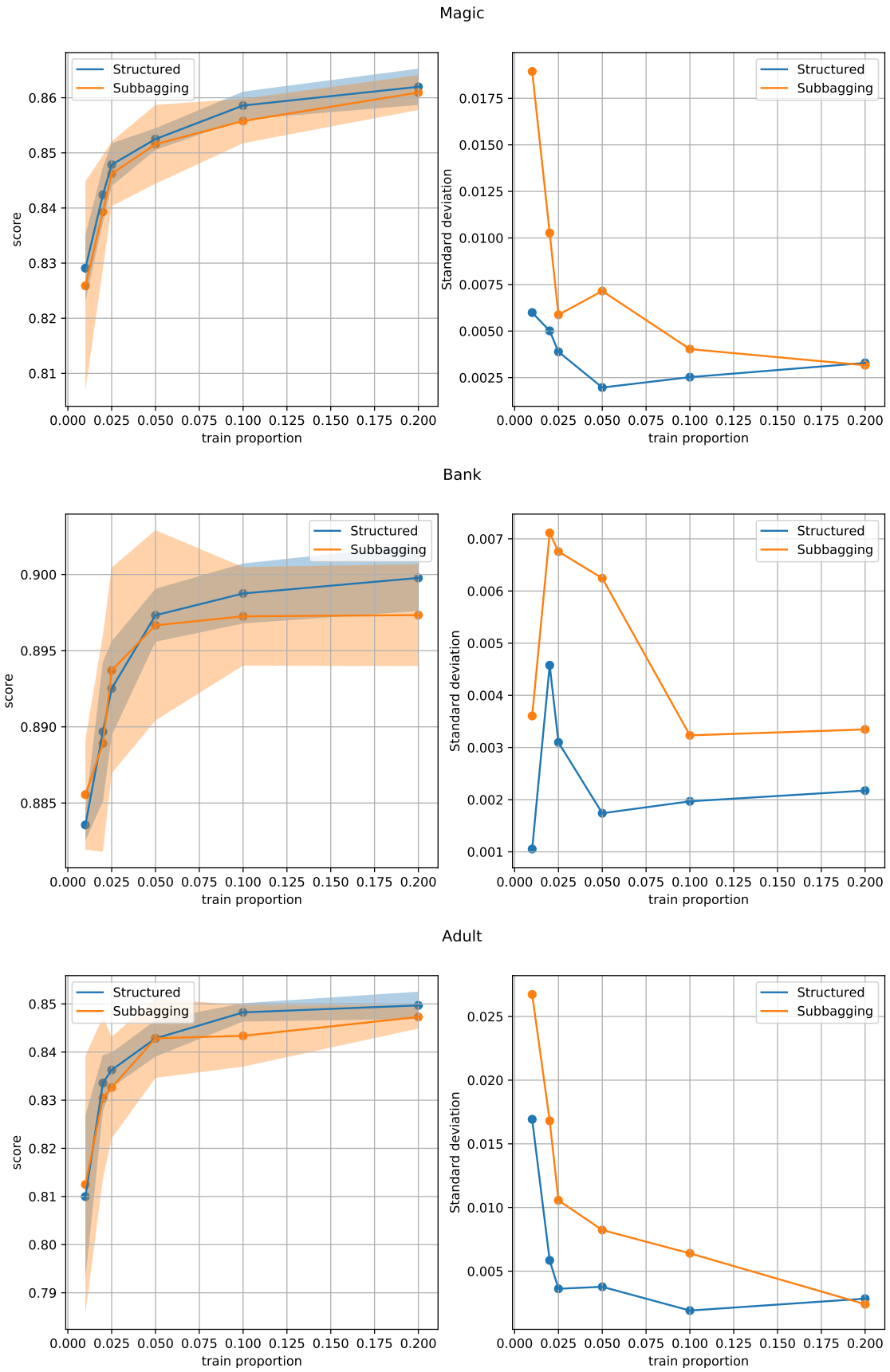

Figure 4.7: Accuracy and robustness comparison between structured and subbagging models in real datasets. The plots in the left represent the average scores of the models. The plots in the left represent their scores standard deviations.

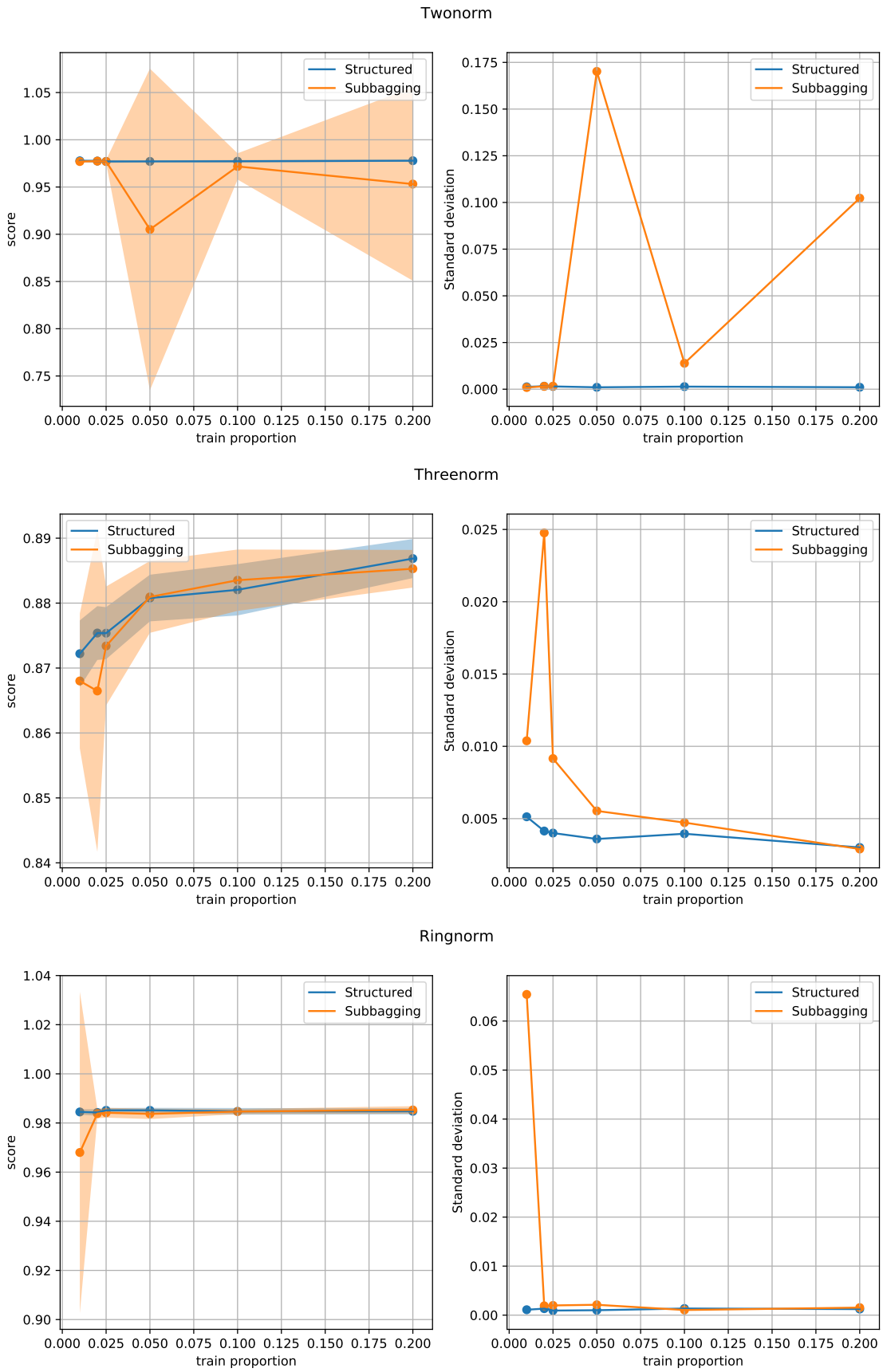

Figure 4.8: Accuracy and robustness comparison between structured and subbagging models in real datasets. The plots in the left represent the average scores of the models. The plots in the left represent their scores standard deviations.

5

Conclusions and future work

In this undergraduate thesis we have proposed and analyzed a SVM ensemble particularly designed for problems with large volumes of data. In this section we will summarize the results of our work and suggest some interesting lines along which it could be continued.

For our model to be practical in problems with large datasets it was of paramount importance that it had a low computational cost. However, accuracy and stability in the predictions should not be hindered in the pursue of this goal. The model proposed in this work is a SVM ensemble built with a special variation of subbagging. This variation is specifically designed to heavily reduce training computational complexity while maintaining accuracy and stability.

The tests carried out show that the proposed ensemble model achieves accuracies comparable to those of a single SVM while significantly reducing training time. For equivalent scores the ensemble reduced training times by a factor of 10 on average and by a factors up to 36 in the best cases. For any given time budget, the proposed model always achieved a better score. In addition, the training cost of the model can be easily adjusted with its parameters B and tp (they in turn regulate the sampling ratio and the size of the ensemble). The comparison with regular subbagging shows that models achieve almost identical accuracies. However, the proposed ensemble shows a substantial improvement in stability, specially for low values of tp, that are related with low sampling ratios.

One of the methods used for building the ensemble was the diversification of the SVM hyperparameters C and γ. This was done by using a completely optimized or exhaustive search over a grid in a sample of the training data. There are other approaches such as random or partially optimized searches of the grid that have been proven to be effective. Exploring the introduction of these methods into the developed ensemble seems to be an interesting follow-up to this work, as it could further increase the speedups in training we have achieved.

APPENDICES

A

Methodology for accuracy test.

In this appendix we provide the pseudocode for the tests presented in Section 4.2. They correspond to the variations for the ensemble of SVM and the single SVM; in synthetic datasets and real datasets.

Input: $N_{repetitions}$ % Number of repetitions
N_{train} % List of train sizes
N_{test} % Size of test set
gen_data % Generates samples of the dataset
Output: $Train\_times$ % Train time statistics for each element of N_{train}
$Accuracies$ % Accuracy statistics for each element of N_{train}

$\mathcal{D}_{test} \leftarrow gen\_data(N_{test})$
for $i \leftarrow 1$ ***to*** $length(N_{train})$ **do**
  for $k \leftarrow 1$ ***to*** N_{reps} **do**
    $\mathcal{D}_{train}^{(i,k)} \leftarrow gen\_data(N_{train}^{(i)})$
    $c^{(i,k)} \leftarrow Single\_SVM(\mathcal{D}_{train}^{(i,k)})$
    $train\_time^{(i,k)}, accuracy^{(i,k)} \leftarrow evaluate(c, \mathcal{D}_{test})$
  $Train_{times}^{(i)} \leftarrow mean(train\_time^{(i,k)}, k), std(train\_time^{(i,k)}, k)$
  $Accuracies^{(i)} \leftarrow mean(accuracy^{(i,k)}, k), std(accuracy^{(i,k)}, k)$
return $Train\_times, Accuracies$

Algorithm A.1: Single SVM test for synthetic datasets.

Input: $N_{repetitions}$ % Number of repetitions
N_{train} % List of train sizes
$\mathcal{D}$ % Dataset
Output: $Train\_times$ % Train time statistics for each element of N_{train}
$Accuracies$ % Accuracy statistics for each element of N_{train}

for $i \leftarrow 1$ ***to*** $length(N_{train})$ **do**
  for $k \leftarrow 1$ ***to*** N_{reps} **do**
    $\{\mathcal{D}_{train}^{(i,k)}, \mathcal{D}_{test}^{(i,k)}\} \leftarrow split\_train\_test(\mathcal{D}, N_{train}^{(i)})$
    $c^{(i,k)} \leftarrow Single\_SVM(\mathcal{D}_{train}^{(i,k)})$
    $train\_time^{(i,k)}, accuracy^{(i,k)} \leftarrow evaluate(c, \mathcal{D}_{test}^{(i,k)})$
  $Train_{times}^{(i)} \leftarrow mean(train\_time^{(i,k)}, k), std(train\_time^{(i,k)}, k)$
  $Accuracies^{(i)} \leftarrow mean(accuracy^{(i,k)}, k), std(accuracy^{(i,k)}, k)$
return $Train\_times, Accuracies$

Algorithm A.2: Single SVM test for real datasets.

Input: $N_{repetitions}$ % Number of repetitions
tp % List of train sizes
N_{test} % Size of test set
gen_data % Generates samples of the dataset
Output: $Train\_times$ % Train time statistics for each element of tp
$Accuracies$ % Accuracy statistics for each element of tp

$\mathcal{D}_{test} \leftarrow gen\_data(N_{test})$
for $i \leftarrow 1$ ***to*** $length(tp)$ **do**
  for $k \leftarrow 1$ ***to*** N_{reps} **do**
    $\mathcal{D}_{train}^{(i,k)} \leftarrow gen\_data(10000)$
    $c^{(i,k)} \leftarrow Ensemble\_SVM(tp^{(i)}, \mathcal{D}_{train}^{(i,k)})$
    $train\_time^{(i,k)}, accuracy^{(i,k)} \leftarrow evaluate(c, \mathcal{D}_{test})$
  $Train_{times}^{(i)} \leftarrow mean(train\_time^{(i,k)}, k), std(train\_time^{(i,k)}, k)$
  $Accuracies^{(i)} \leftarrow mean(accuracy^{(i,k)}, k), std(accuracy^{(i,k)}, k)$
return $Train\_times, Accuracies$

Algorithm A.3: Ensemble SVM test for synthetic datasets.

Input: $N_{repetitions}$ % Number of repetitions
tp % List of tps
$\mathcal{D}$ % Dataset
Output: $Train\_times$ % Train time statistics for each element of tp
$Accuracies$ % Accuracy statistics for each element of tp

for $i \leftarrow 1$ ***to*** $length(tp)$ **do**
  for $k \leftarrow 1$ ***to*** N_{reps} **do**
    $\{\mathcal{D}_{train}^{(i,k)}, \mathcal{D}_{test}^{(i,k)}\} \leftarrow split\_train\_test(\mathcal{D}, 10000)$
    $c^{(i,k)} \leftarrow Ensemble\_SVM(tp^{(i)}, \mathcal{D}_{train}^{(i,k)})$
    $train\_time^{(i,k)}, accuracy^{(i,k)} \leftarrow evaluate(c, \mathcal{D}_{test}^{(i,k)})$
  $Train_{times}^{(i)} \leftarrow mean(train\_time^{(i,k)}, k), std(train\_time^{(i,k)}, k)$
  $Accuracies^{(i)} \leftarrow mean(accuracy^{(i,k)}, k), std(accuracy^{(i,k)}, k)$
return $Train\_times, Accuracies$

Algorithm A.4: Ensemble SVM test for real datasets.

Bibliography

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, Sep 1995.

[2] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, Jun 1998.

[3] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, Aug 1996.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[6] R. A. FISHER, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[7] Z. Yong, L. Youwen, and X. Shixiong, "An improved knn text classification algorithm based on clustering," *Journal of Computers*, vol. 4, 03 2009.

[8] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support vector machines for histogram-based image classification," *IEEE Transactions on Neural Networks*, vol. 10, pp. 1055–1064, Sep. 1999.

[9] S. Knerr, L. Personnaz, and G. Dreyfus, "Handwritten digit recognition by neural networks with single-layer training," *Neural Networks, IEEE Transactions on*, vol. 3, pp. 962 – 968, 12 1992.

[10] M. Friedl and C. Brodley, "Decision tree classification of land cover from remotely sensed data," *Remote Sensing of Environment*, vol. 61, pp. 399–409, 09 1997.

[11] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[12] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: a new explanation for the effectiveness of voting methods," *Ann. Statist.*, vol. 26, pp. 1651–1686, 10 1998.

[13] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, Mar. 2002.

[14] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, (New York, NY, USA), pp. 144–152, ACM, 1992.

[15] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012.

[16] T. G. Dietterich, "Ensemble Methods in Machine Learning," *Lecture Notes in Computer Science*, vol. 1857, pp. 1–15, 2000.

[17] K. Ali, "On the link between error correlation and error reduction in decision tree ensembles," 1995.

[18] D. M. Estlund, "Opinion leaders, independence, and condorcet's jury theorem," *Theory and Deci-*

sion, vol. 36, pp. 131–162, Mar 1994.

[19] P. Bühlmann and B. Yu, "Analyzing bagging," *Ann. Statist.*, vol. 30, pp. 927–961, 08 2002.

[20] R. K. Bryll, R. Gutierrez-Osuna, and F. K. H. Quek, "Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets," *Pattern Recognition*, vol. 36, pp. 1291–1302, 2003.

[21] G. Martínez-Muñoz and A. Suárez, "Switching class labels to generate classification ensembles," *Pattern Recognition*, vol. 38, no. 10, pp. 1483–1494, 2005.

[22] R. E. Schapire, "A brief introduction to boosting," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, (San Francisco, CA, USA), pp. 1401–1406, Morgan Kaufmann Publishers Inc., 1999.

[23] J. Stork, R. Ramos, P. Koch, and W. Konen, "Svm ensembles are better when different kernel types are combined," in *Data Science, Learning by Latent Structures, and Knowledge Discovery* (B. Lausen, S. Krolak-Schwerdt, and M. Böhmer, eds.), (Berlin, Heidelberg), pp. 191–201, Springer Berlin Heidelberg, 2015.

[24] B. Parmanto, P. W. Munro, and H. R. Doyle, "Improving committee diagnosis with resampling techniques," in *Advances in Neural Information Processing Systems 8* (D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, eds.), pp. 882–888, MIT Press, 1996.

[25] R. P. W. Duin and D. M. J. Tax, "Experiments with classifier combining rules," in *Multiple Classifier Systems*, 2000.

[26] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241 – 259, 1992.

[27] A. Buja and W. Stuetzle, "Observations on bagging," *Statistica Sinica*, vol. 16, no. 2, pp. 323–351, 2006.

[28] G. Martínez-Muñoz and A. Suárez, "Out-of-bag estimation of the optimal sample size in bagging," *Pattern Recogn.*, vol. 43, pp. 143–152, Jan. 2010.

[29] V. Cherkassky and Y. Ma, "Practical selection of svm parameters and noise estimation for svm regression," *Neural Networks*, vol. 17, no. 1, pp. 113 – 126, 2004.

[30] M. Sabzevari, G. Martínez-Muñoz, and A. Suárez, *Randomization vs Optimization in SVM Ensembles: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part II*, pp. 415–421. 09 2018.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, and G. Louppe, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, 01 2012.

[32] L. Breiman, "Bias, variance, and arcing classifiers," Tech. Rep. 460, Statistics Department, University of California, 1996.

[33] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[34] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22 – 31, 2014.